Lila: Optimal Dispatching in Probabilistic Temporal Networks using Monte Carlo Tree Search

Michael Saint-Guillain[1], **Tiago S. Vaquero**[2], **Steve A. Chien**[2]
[1] Université catholique de Louvain, Place de l'Universite 1, 1348 Ottignies-Louvain-la-Neuve, Belgium
[2] Jet Propulsion Laboratory, California Institute Technology, 4800 Oak Grove Dr, Pasadena, CA 91109 USA
michael.saint@uclouvain.be, {tiago.stegun.vaquero, steve.a.chien}@jpl.nasa.gov

Abstract

Executing a Probabilistic Simple Temporal Network (PSTN) amounts at scheduling, i.e. *dispatch*, a set of events under time uncertainty. This constitutes a *NP*-hard online optimization problem. The right execution time must be dynamically assigned to each event of the PSTN such that the temporal constraints are met, whereas activity durations are progressively observed as the execution unfolds. We propose a dispatching algorithm based on Monte Carlo Tree Search, called *Lila*, with the following characteristics: (i) it is an anytime algorithm, both offline and online, conjectured asymptotically optimal; (ii) it returns the current probability of success, either before or at any moment during operations; (iii) it handles any possible continuous or discrete, even non-parametric, probability distributions, as well as interdependencies between random variables, exogenous and endogenous uncertainty; and (iv) can be easily extended to handle probabilistic external events, PSTNs with resources, PSTNs with cutoff times and precondition chains, *etc.* Lila is universal in the sense that it can handle any dispatching protocol, simply by specifying it to the algorithm. It has the unlimited flexibility offered by the simulation paradigm, and is conjectured to asymptotically converge to optimal decisions and/or robustness approximations.

1 Introduction

Temporal networks formalize the arrangement and interdependencies of tasks, or activities, that compose an operational plan. In a simple temporal network (STN), activities are modelled as a finite set of time events, such as start and end times. In practice, some activity durations, considered as *contingent*, remain unknown beforehand and are revealed during execution (decided by nature). When some stochastic knowledge on the uncertain durations exists, one can model it as (estimated) probability distributions, leading to the extending concept of probabilistic STN, or *PSTN*. Solving a PSTN then amounts at finding an assignment of time values to executable events, such that assigned values together with observed ones fulfil all the constraints between events (*e.g.*, end of task A must happen between 10 and 20 minutes before the beginning of B). Whenever such assignment exists, a network is said to be *controllable*. When the operational assumptions enable it, the assignment may be *dynamically constructed*, i.e. as durations are observed, the time values

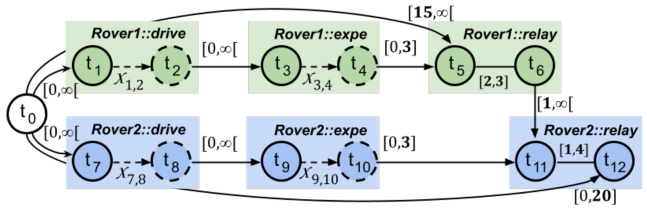

Figure 1: A simplified hypothetical sol on Mars for two planetary rovers, encoded as a PSTN. Bold: controllable. Dashed: contingent.

are assigned. Yet, even under dynamic decision, due to unfortunate durations, a network may reveal as violating some of the temporal constraints during execution.

When there is uncertainty with respect to temporal constraint violation, it is critical to determine the actions (*i.e.* assign time values to time events) that maximize the probability of successful execution. Furthermore, at a given state of the online execution, determining the current probability of success according to past decisions and events as well as the remaining uncertainty is a major importance too. In case this probability falls below some threshold of acceptable risk, the operators may decide to interrupt the execution before it reaches a problematic or dangerous state.

Fig. 1 shows an hypothetical example of Mars rovers operations as a PSTN. Each rover has three activities in sequence: drive towards a science site, perform a science experiment, and relay results to an orbiter. A special time point $t_0 = 0$ represents the beginning of the operations. Time events are linked by temporal constraints, either controllable or contingent. The rovers work independently during their driving and science activities, and eventually coordinate for the communication time window, which strictly happens within time 15 to 20. Furthermore, a maximum of 3 time units is authorized from the end of experiments to the start of relay activities, implying that starting everything as soon as possible may be problematic. The duration of the driving and science activities are uncertain, and encoded in the PSTN as contingent constraints described by probability distributions. Ideally, a perfect assignment of all executable time points would work for any situation imposed by nature. In practice that is very restrictive, if not impossible, especially in highly uncertain environments. It raises the following questions. What is the probability that we succeed at executing the PSTN, namely that our rovers both meet the communication window? How to compute online decisions in an optimal way, such that we maximize this probability?

Contributions. We describe how the MCTS framework can be used in the context of scheduling time points in constrained temporal networks under uncertainty, namely PSTN dispatching. To the best of our knowledge, this is the first application of MCTS to temporal networks. Unlike mathematical approaches, or even those based on pure Monte Carlo simulation, our method is an anytime algorithm; it is conjectures as asymptotically optimal (we let the demonstration for future work), and allows to consider various extensions to the classical PSTNs with minor adaptations. Moreover, MCTS allows to simply handle very complicated concepts, some of them being described in this section, such as dependent random variables or even endogenous uncertainty (which is a topic rarely covered in the literature). A preliminary experimental analysis is conducted.

2 Probabilistic Simple Temporal Networks

Simple Temporal Network is a popular formalism for temporal constraint reasoning (Dechter, Meiri, and Pearl 1991), framed as a constraint satisfaction problem over time point variables: a STN is a tuple $\langle T, C\rangle$, where T is a set of time points and C is a set of constraints $c(t_i, t_j)$ that encode bounds on the differences between pairs of time points: $l_{ij} \leq (t_j - t_i) \leq u_{ij}$, *i.e.* $(t_j - t_i) \in [l_{ij}, u_{ij}]$. The goal is then to assign time values to every time points, such that all the $t_j - t_i$ duration constraints are respected.

Most realistic operational contexts account for temporal uncertainty. **PSTN** is a natural extension of STN in which probability density functions are associated to temporal constraints, such as activity durations (Tsamardinos 2002). In a PSTN, the *executable* time points T_E are determined by the agent, and *contingent* time points T_C are assigned by nature. A solution is called a *schedule*, a specific assignment to all $t_i \in T_E$. Given a particular realization of T_C, a schedule is *consistent* if it satisfies all the constraints of the network. In practice a contingent duration is described by a (usually estimated) probability distribution $(t_j - t_i) = X_{i,j}$.

In practice, a time point in the PSTN often stands for either the start (*e.g.* t_3 in Fig. 1) or the end (t_4) of a particular *activity* (*Rover1::expe*). The starting point of activities usually constitute the set of executable time points T_E. A schedule determines the execution time of $t_j \in T_E$, and requirement constraints in the form $c(t_i, t_j)$ state how late t_j can occur regarding to any previous time points t_i. When $t_j \in T_C$, which could represent an activity completion, the duration $(t_j - t_i)$ remains unknown prior to execution.

Policies and Dispatching protocol. Operational contexts such as space missions usually pose computational and power limitations on recomputing a schedule in the middle of the operations (Chi et al. 2019). Yet, the use of a static schedule is often either impossible in practice, or comes with a significant waste in terms of operational yield and time. Such approach is currently operating *Curiosity* rover, with static schedules that overestimate processing times by 30% in average (Gaines et al. 2016) to account to execution uncertainty. Let Ω be the set of all possible realizations of the random contingent edges' duration in the PSTN. A trivial approach to avoid both static scheduling and online reoptimization is to precompute particular schedules for each possible situation that may arise, leading to a *policy*. Naturally, the size of Ω is usually problematic. Instead, Perseverance (M2020) rover is equipped with a non-backtracking onboard scheduler, designed to take online decisions based on current observations (Rabideau and Benowitz 2017; Chi et al. 2018; Agrawal et al. 2021a,b). Due to computational limitations, such online decisions must remain very light, thus following a predefined *strategy*: a *dispatching protocol* (DP). In particular, a DP usually aims at avoiding costly online reoptimizations. For example, Rabideau and Benowitz (2017) describe an average $\mathcal{O}(n^2)$ quadratic DP$(\cdot)$ protocol to be computed by the onboard scheduler in the Mars Perseverance rover, in order to adapt decisions online (*i.e.* $\Gamma_E^t = T_E^t$) based on observations and pre-optimized parameters (Chi et al. 2019).

***NextFirst* dispatching protocol.** The *NextFirst* protocol (Brooks et al. 2015), also known as *DC-dispatch* (Morris, Muscettola, and Vidal 2001) or *early execution* (Lund et al. 2017), dynamically assigns a value to and dispatches each time point (*i.e.* executes the PSTN) in $\mathcal{O}(n)$ linear time, by starting activities as soon as possible. Let t_j be a controllable time point in a PSTN, and $I_j = \{(0, j), \ldots, (i, j)\}$ the set of incoming edges in t_j. Therefore, t_j is assigned a time value as soon as all the preconditions are validated, that is, all the $t_0, \ldots, t_i$ time points are known, leading to the very simple online decision rule:

$$t_j = \max(t_0 + l_{0j},\ \ldots\ , t_i + l_{ij}). \qquad (1)$$

In the case $t_j > \min(t_0 + u_{0j},\ \ldots\ , t_i + u_{ij})$, the dynamic execution is interrupted and considered as failed. Naturally, *NextFirst* protocol has linear complexity $\mathcal{O}(n)$. Back to our PSTN example in Fig. 1, the value of t_{11} is then dynamically set to $\max(t_{10},\ t_6)$ as soon as tasks *Rover2:expe* and *Rover1:relay* are completed. Execution fails if t_{11} exceeds $t_{10} + 3$. Eventually, we hope for $t_{12} \leq 20$.

Formulation of the optimal dispatching problem

Assumptions and notations. We assume the operational time horizon to be partitioned in h outcome and decision stages. The random vector $\xi = \xi^1, \ldots, \xi^h$, with support Ω, describe all the possible sequences of outcomes. When necessary, we designate by $\xi^{t..t'}$ the sequence of outcomes of scenario ξ from time t to time t'. The decisions to be taken by the online scheduler during the operations are represented by a vector $\mathbf{x} = x^1, \ldots, x^h$ of $\mathbb{R}^h$, from which the schedule can be trivially deduced. We refer to decisions $x^t, \ldots, x^{t'}$ as $x^{t..t'}$. The indicator function $V^{\mathbf{x}}(N, \xi)$ returns 1 iff the schedule $\mathbf{x}$ is consistent in scenario ξ. Operator $E_{\xi^t}[\ \cdot\ ]$ designates the expectation over random variable ξ^t, conditionally to history $\xi^{1..t-1}$. In case of endogenous uncertainty,

$E_{\xi^t}[\,\cdot\,]$ also depends on decisions $x^{1..t-1}$. Finally, X^t represents the set of legal actions (*i.e.* time assignments) at time t, which naturally depends on past actions $x^{1..t-1}$ and history $\xi^{1..t}$.

Multistage stochastic formulation. An optimal dispatching protocol necessary computes, in any possible situation (*i.e.* given any possible past realizations and decisions), the decisions that maximizes the probability that the current partial schedule completes to a consistent schedule. The following multistage stochastic program determines the optimal dispatching decisions x^t at a current time t:

$$\underset{x^t \in X^t}{\operatorname{argmax}} \; E_{\xi^{t+1}}\Big[\max_{x^{t+1}\in X^{t+1}} E_{\xi^{t+2}} \Big[\dots \max_{x^{h-1}\in X^{h-1}} E_{\xi^h}\Big[\max_{x^h \in X^h} V^{x^{1..h}}(N,\xi)\Big] \dots \Big]\Big] \quad (2)$$

where the maximum value of the first expectation, when $t=0$, is by definition equal to the Degree of Dynamic Controllability (DDC) of the network (Saint-Guillain et al. 2020, 2021), the probability of success under perfect reoptimization. Consequently, this must be at least equal to the probability of success under the *NextFirst* protocol.

The nested expectations in (2) form a tree structure, well known as the *scenario tree*. Unfolding the maximization operators as well leads to a full decision-scenario tree as illustrated in Fig. 2. Each path of the tree constitutes a possible scenario realization together with associated decisions, a sequence $\xi^t, x^t, \dots, \xi^h, t^h$. At time t, decisions x^t depend on the current history $\xi^{1..t}$ and maximize the expected value $E_{\xi^{t+1}}[\max_{x^{t+1}} \dots]$ of the future decisions at time $t+1$ given the remaining uncertainty, and so on until time h is reached.

A node ξ^t hence represents a particular state, defined by history $\xi^{1..t}$ and past decisions $x^{1..t-1}$. Counter-intuitively, we call node ξ^t a *decision node*. This is because at this particular state, a decision x^t must be chosen amongst X^t. Following (2), x^t is necessarily the decision maximizing the probability that the partial schedule $x^{1..t}$ extends to a consistent full schedule.

A node x^t represents a decision that has already been chosen for time t. Since it directly leads to nodes representing the possible realizations of ξ^{t+1}, we call x^t a *chance node*. Still following (2), the value of the decision x^t is given by the expected value of the subsequent (and consequent) realizations.

3 Monte Carlo Tree Search

Solving problem (2) is computationally intractable in practice. Yet, a look at the associated tree immediately suggests two classical approximation schemes: (a) limiting the branching factor and (b) avoiding to consider less relevant subtrees. Both approaches are compatible, and with a few additional techniques described in this section, we will end up with an adaptation of the well-known Monte Carlo Tree Search (MCTS) algorithm to our PSTNs.

Limiting the branching factor. This can be achieved by sampling a restricted number of children generated from

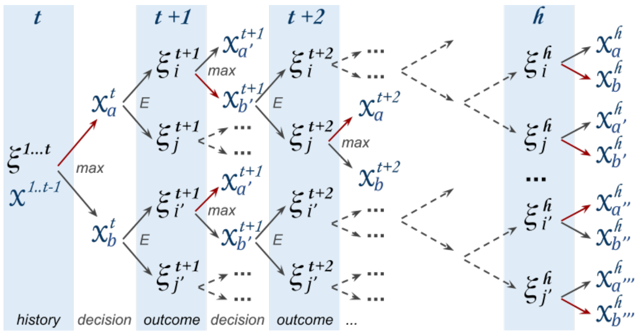

Figure 2: Tree structure of the problem. The root node represents the current state (past decisions and realizations) at time t. For simplicity, decision (*resp.* random) variables have only two possible choices (*resp.* outcomes).

chance and/or decision nodes. At a decision node, a limited number of children $x_a^t, x_b^t, \dots$ could either be chosen in X^t at random, or by following some predefined strategy (a dispatching protocol!). At a time t chance node, a limited sample of random realizations of ξ^{t+1} may constitute a pertinent restricted set of children nodes. In particular, even if MCTS aims at dealing with large branching factors, this step is still mandatory to obtain a discrete tree from the decision and realization domains, which are continuous by nature.

Subtrees prioritization. While a huge part of the whole tree depicted in Fig. 2 is already pruned by the simple action of restricting the branching factor, the resulting tree is likely to remain too big to be entirely explored, and further decreasing the branching factor may result in missing critical decisions or outcomes. This is where MTCS comes at hand. In fact, MCTS selects the most promising nodes (*i.e.* subtrees) to consider first by using a node value function, which exploits a Monte Carlo sampling paradigm to approximate and eventually evaluate the interest in visiting a subtree.

General MCTS approach and related work

MCTS is a general framework, which has already been well described by the literature. The reader interested in a complete description may refer to Browne et al.'s survey (2012), Section III. The general MCTS algorithm can be outlined as:

MCTS keeps iteratively performing each of the following four steps in turn, until the computation budget is reached: **1) select** an expandable (non final) node; **2) expand** the node by generating one (or more) of its child nodes; **3) simulate** one path down the current tree, in order to reach a final state from the newly created child node, without creating any new node, but instead obtain a final state as quickly as possible; **4) backpropagate** the final state value R, updating the estimated value $V(\text{node})$ of each node along the path from the new to the root node. Finally, the playing action represented by the best root child is returned once the computational budget is exhausted. An example of search tree being gradually built using MCTS's four steps is depicted in Figure 3. Some details of the tree may appear mysterious at

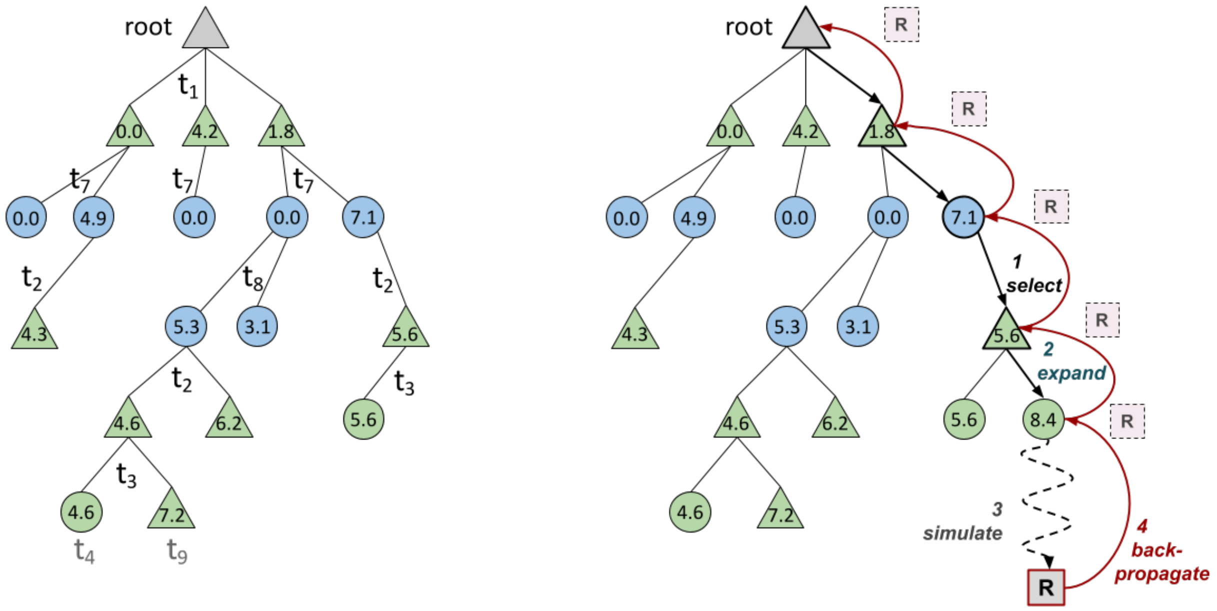

Figure 3: MCTS tree and iteration steps. △ Decision node. ○ Chance node. □ Terminal node.

Algorithm 1: General MCTS.

```
1  root ← CreateRootNode();
2  while computation budget not exhausted do
3      node ← root;
4      while not node.IsExpandable() do
5          node ← node.SelectChild();
6      child ← node.ExpandOne();
7      if child.IsTerminal() then
8          child.BackPropagate(child.Evaluate());
9      else
10         for $i$ ← 1 to $n_{sim}$ do
11             child.BackPropagate(child.Simulate());
12 return root.BestChild();
```

this time, and will be clarified as we describe how we adapt MCTS specifically for taking decisions in PSTNs.

How to select which node to expand, and therefore where to grow the tree, is at the heart of MCTS and raises the question of intensification versus diversification. Whereas a selection policy based on intensification only is likely to end up exploring a very narrow, specific, deep subtree, the opposite pure diversification would result in simple breadth-first search. The most popular selection policy, called Upper Confidence Bounds for Trees (UCT), makes its path from the root node down to a leaf node to be expanded (*i.e.* not a final game state) by diving through child nodes (lines 3-5 of Algorithm 1), where ***node.SelectChild()*** maximizes

$$UCT = V(\text{child}) + 2C\sqrt{\frac{2\ln n^{\text{node}}}{n^{\text{child}}}}$$

where the first term $V(\text{child})$ is the estimated value of the child node and therefore encourages intensification. The second term, with n^{node} being the number of times a node or one of its descendants ran a simulation, encourages diversification by promoting children being less visited than their sibling nodes. In fact, a never visited child will be given ∞ value. The C parameter is usually empirically tuned to best calibrate both terms. ***root.BestChild()*** classically returns the root child maximizing either $V(\text{child})$ or n^{child}. Remark that whereas $V^{x^{1..h}}(N, \xi)$ in Eq. (2) indicates whether a leaf node of the decision/outcome tree is a win or a loss state, here $V(\text{node})$ approximate the expected success value of a terminal node belonging to the subtree defined by $node$.

MCTS with continuous action and outcome spaces. The classical MCTS method has been developed for deterministic zero-sum games. Like stochastic games that involve rolling a dice, our PSTNs involve uncertainty as the contingent duration of some activity remains unknown until one actually tries to execute it. Naturally, MCTS have been adapted to deal with dice rolls, and variants have been proposed (Browne et al. 2012; Cowling, Powley, and Whitehouse 2012). However, most studies focus on partially observable states, such as hidden cards in Poker, rather than uncertainty. As PSTNs usually involve a continuous time dimension, both realization outcomes and action decisions must generally be chosen out of continuous domains. Different methods have already been proposed in order to obtain a discrete search space compatible with MCTS, namely to implement functions ***node.isExpandable()*** and ***node.ExpandOne()***. Amongst the proposed approaches, many are based on sampling a limited set of actions and/or outcomes (Kearns, Mansour, and Ng 2002), that progressively grows as the associated node is being visited (Chaslot et al. 2008; Couëtoux et al. 2011), a technique called *progressive widening*. In fact, *Lila* exploits the progressive widening technique. Back to the ***node.SelectChild()*** function, Yee, Lisỳ, and Bowling (2016) proposed KR-UCT, an alternative to UCT based on kernel regression, to better select and further share information between actions sampled from continuous domains. They apply KR-UCT on the remarkable problem of autonomous agents playing *curling*.

Dispatching a PSTN: a single-player game against Nature

In a sense, our approach suggests to model our PSTN as a single-player game against Nature. At a given state, the player's actions aim at choosing whether, for each of the time events that are ready for execution, to schedule them immediately or postpone. Depending on the player's decisions, dice rolls will then be used to represent the possible random completion times of each started activity. Ideally, the search tree being iteratively built by the MCTS algorithm should directly approximate the full decision-scenario tree depicted in Fig. 2. In practice however, depending on the live decisions and outcomes, most of the time units in $t..h$ involve no decision nor outcome.

An equivalent, *event-driven* tree can be obtained by simply skipping all "empty" time units. Furthermore, this approach permits to get rid of discrete time units and to handle a continuous time horizon. The MCTS tree depicted in Fig. 3 gives an example of such construction, for our rover PSTN example of Fig. 1. Here, the execution has not yet started (*i.e.* current real time is zero) and MCTS is used to approximate the full decision-scenario tree of our PSTN, beginning with the decisions of when to schedule events t_0 and t_7. Recall that decision nodes ($\triangle$) stand for a state where the upcoming action is a decision, and chance nodes ($\bigcirc$) states involve a pending random outcome.

A path of our MCTS tree does not simply alternate decision and chance nodes. In fact, the nature of the node, and even the action being played, depends on the history (path) rather than its depth. For instance, the node $t_3 = 4.6$ at the bottom left is a chance node ($\bigcirc$), whereas its sibling $t_3 = 7.2$ is a decision node ($\triangle$). This is because in both cases the history is $t_1 = 1.8, t_7 = 0, t_8 = 5.3, t_2 = 4.6$. If t_3 is to be scheduled directly after t_2 at time 4.6, then the upcoming event is the random outcome t_4, since t_8 is only at 5.3. If on the contrary t_3 is delayed to 7.2, then the next event concerns the action of deciding for t_9.

Proposed PSTN-specific MCTS instantiation

We now describe how we currently propose to instantiate Algorithm 1 in order to obtain a PSTN (online), asymptotically optimal, dispatching algorithm.

node.IsExpandable() When executing a PSTN, both decisions and outcomes must be selected from (continuous) infinite domains. Similarly to Kearns, Mansour, and Ng (2002), we experiment a fixed-size branching factor at both chance and decision nodes. Therefore, the function returns true iff the predefined size is not yet reached. We also consider a more elaborated strategy, designing *node.IsExpandable()* such that the branching factor of a node progressively increases (Chaslot et al. 2008; Couëtoux et al. 2011), hence allowing the asymptotic completeness of the algorithm, that is, the optimally robust decisions at defined by Eq. (2). The branching factor of each node then depends on how many times it has been visited, and follows βn^α, with $n = n^{\text{node}}$.

node.ExpandOne() At a chance node, a child is created simply by sampling the associated random variable following its own probability distribution. At a decision node, different approaches may be considered when selecting an execution time. In order to eventually converge to a complete search tree, any relevant execution time should be possibly chosen. In this paper, the first generated child is executed as soon as possible according to the PSTN lower bounding time constraints, without any delay. Any other child gets assigned an execution times randomly sampled, with a probability that decreases with the delay. This is achieved by taking the absolutes values from a normal distribution centred at delay 0.

Note that if the decision nodes are limited to have only one child, then the MTCS tree will naturally converge to represent the behavior of the PSTN under the *NextFirst* dispatching protocol, which consists in executing each time point as soon as possible. In theory, the *node.ExpandOne()* can therefore be implemented in order to represent any computable dispatching protocol in MCTS. However, in order to get completeness, any admissible execution delay should be eventually considered.

node.isTerminal() A node is terminal as soon as either all the time points of the PSTN (either associate to chance or decision nodes) have been attributed a value, or if the value of some time point is not consistent with the PSTN temporal constraints. In practice, an inconsistency may be detected earlier, by comparing the current time assignments with the remaining possible future decisions and outcomes, therefore allowing to avoid further exploring a subtree which can be proven to be inconsistent. For that we refer to established theoretical results on PSTN controllability checking, and leave the related improvements for future work.

node.Simulate() The classical MCTS approach for simulating the remaining decisions and outcomes would consist in sampling everything at random, until a final state is reached. In the specific context of PSTNs however, it has been observed that executing time points as soon as possible provides good results in general (Saint-Guillain et al. 2020). Therefore, our approach is to follow the *NextFirst* dispatching protocol during simulations.

node.BackPropagate() Once the simulation hits a terminal state, its success value (0 or 1) must be backpropagated from the child node that initiated the simulation, up to the root node, thereby updating the estimated values of all the nodes along that path. The "expectiminimax" rule (Melkó and Nagy 2007) is applied, yet adapted to a single player stochastic game: if the node n is terminal, then its value $V(n)$ is equal to the success value; if it is a decision node, then its value is updated to $V(n) = \max_{c \in \text{Children}} V(c)$; if it is a chance node, then $V(n) = \frac{1}{|\text{Children}|} \sum_{c \in \text{Children}} V(c)$. Remark that this rule eventually converges to the optimal decision/outcome tree formulated in Equation (2).

root.BestChild() At a current time t, the root node is usually a decision node. Once the computational budget is reached, we are interested in the best possible decisions, below the root node. As shown in Fig. 3 however, a path of our MCTS tree does not simply alternate decision and chance nodes. If, as for the PSTN of Fig. 1 and the associated MCTS tree of Fig. 3, two decisions (t_1 and t_7) follow the root node, then the best two decisions must be returned. In our example, *BestChild()* is called at root node to select the best direct child for t_1, and then is called in turn on that child to select the best subsequent decision for t_7. Amongst the possible choices, we simply select the child maximizing $V(n)$.

4 The Versatility of the Simulation Paradigm

A basic Monte Carlo simulation could not converge to optimal decisions, because any simulation requires to follow a predefined, often simplistic, execution strategy (dispatching protocol) such as *NextFirst* — otherwise each simulation amounts at solving the NP-hard multistage optimization problem, which the simulation aims at approximating! In other words, running a Monte Carlo simulation forever simply converges to the expected value of the predefined strategy. On the contrary, Monte Carlo Tree Search combines Monte Carlo simulation with the construction of a complete, optimal, decision tree, which ultimately (*i.e.* asymptotically) converges to the true optimal decisions. Thanks to the simulation side, MCTS allows to simply handle very complicated concepts, some of them being described in this section, such as dependent random variables or even endogenous uncertainty (which is a topic rarely covered in the literature).

Maximizing the Expected Utility: dealing with Cutoffs and Precondition chains

A classical PSTN assumption is that the PSTN execution fails as soon as an activity is failed at being executed within the time constraints. In Saint-Guillain et al. (2020), we proposed PSTNs alternative execution assumptions, in which activities can be safely interrupted, using a predefined deterministic *cutoff time* or *duration*, hence allowing the execution to continue. In our rover example, this could be true for any experimental activity which are somewhat isolated.

Nonetheless, interrupting an activity may however turn impossible to carry out a related subset of remaining ones, such as for example, an experiment composed of several tasks. In our example of Fig. 1, interrupting a driving activity would necessarily compromise the associated experiment, although it does not prevent from further relaying. In other words, the driving activity is a *precondition* for the subsequent experiment. In practice, precondition chains may span over multiple subsequent activities: if a task C depends on successful execution of B, which in turn depends on a task A, then interrupting A would prevent from executing both B and C.

Yet, all activities do not necessarily have the same priority, and some may even be considered mandatory (uninterruptible), such as the relay activities in our example. A utility value can therefore be assigned to interruptible activities, leading to the objective of maximizing the overall *expected utility* of the network, that is, the expected sum of the task utilities that can be successfully dispatched, whereas failing at dispatching a mandatory activity results in a zero utility. More generally, mandatory tasks may be assigned a very high utility value *w.r.t.* interruptible ones.

MCTS with Utility, Cutoffs and Preconditions. Our PSTN specific instantiation of the MCTS framework is able to deal with these new concepts with a few straightforward adaptations. The *node.IsTerminal()* and *node.ExpandOne()* functions are impacted. A node is then terminal when either an inconsistency is detected (or if it reached its predefined cutoff), or when all time points have been attributed a value. At deciding for the execution time of a task in *node.ExpandOne()* function, the resulting time then never exceeds the predefined cutoff. Furthermore, depending on the history, the task at stake will not be executed (*i.e.* assigned duration zero) if some of its preconditions is not met, such as a past required activity that has been interrupted by hitting its cutoff (or not executed for similar reasons). Finally, the computation of $V(n)$ at a terminal node is not zero or one anymore, but the sum (over the MCTS tree path) of the utilities of the tasks that did not hit their cutoff time, or zero if some mandatory task did. The resulting $V(n)$ is then backpropagated the exact same way as aforementioned. Eventually and following Eq. (2), the average value at $V(root)$ will necessarily converge to the *expected total PSTN utility*.

Dealing with resources and exotic probabilities. Other PSTN extensions, such as resource usage, can be handled by adapting the *node.ExpandOne()* function, assuming that *node.Simulate()* uses the same mechanism to sample decisions and outcomes. One just need to save the current resource usage state, such as energy consumption, at each node of the MCTS tree and deduce, at the current decision node, the possible children accordingly. When it comes to chance nodes, because the children are simply randomly sampled from each random variable distribution, any possible distribution may be considered. In fact, considering the current history of decisions and outcomes at a certain chance node, dealing with dependent random variables (*i.e.* possible outcomes being influenced by past realizations), as well as endogenous uncertainty (*i.e.* influenced by past decisions) becomes just a matter of implementing the *node.ExpandOne()* function.

5 Experimental analysis and validations

The rover PSTN example of Fig. 1 constitutes an interesting benchmark for analyzing the behavior of the proposed framework. Since computing the probability of success of the optimal decisions (*i.e.* the Degree of Dynamic Controllability as defined in Saint-Guillain et al., 2020) is intractable,

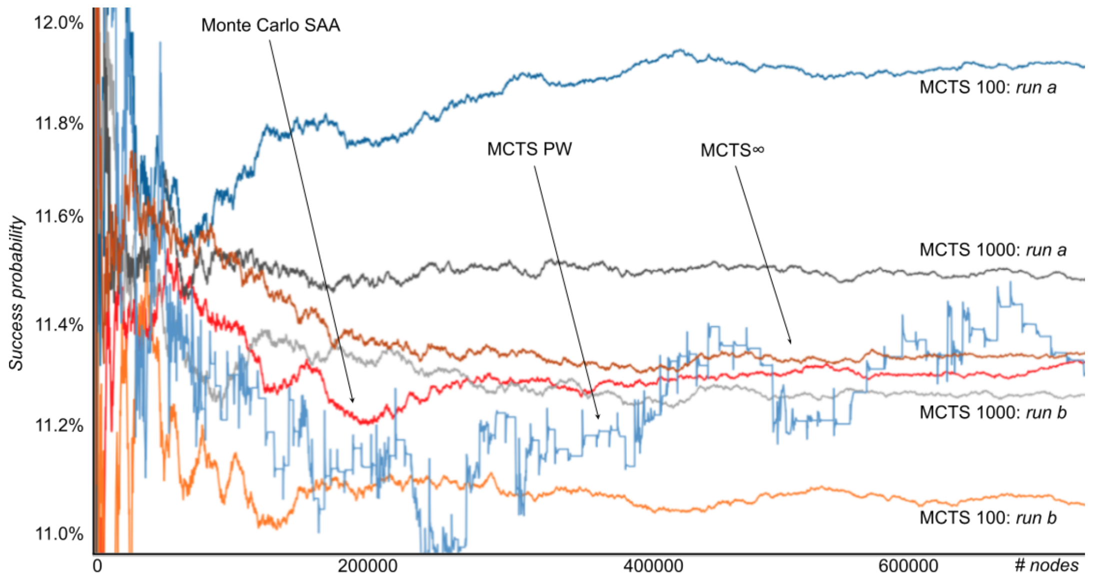

Figure 4: Evolution of the $V(root)$ estimated root node value, which predicts the robustness of the network under *NextFirst*, since decision nodes are limited to one child, for PSTN example of Fig. 1. A basic Monte Carlo simulation, in red, converges to the true robustness value.

there is the need for approximation methods such as the one proposed here. In this case, we must rely on other indicators to empirically validate our method. We then first consider the robustness of the PSTN under the *NextFirst* dispatching protocol, which can be either computed exactly, or approximated with an arbitrary precision using Sample Average Approximation (SAA), namely pure Monte Carlo simulation. Thereafter, more elaborated dispatching decisions will be considered by allowing Lila to try postponing the beginning of PSTN activities, leading to an approximation of the true probability of success under perfect reoptimization.

Implementing NextFirst dispatching protocol. Given some parametrization that restricts Lila's decisions to follow *NextFirst* protocol, the value of $V(root)$ node should converge to the robustness under *NextFirst*. We hence limit the number of decision node children to one, while varying the number of chance node children. As explained with *node.ExpandOne()* function, the resulting MCTS tree should then approximate the behavior of the PSTN under *NextFirst*.

Figure 4 shows how our $V(root)$ estimated root node value converges compared to the the success rate measured by SAA (Monte Carlo), as the number of iterations (*i.e.* nodes for MCTS) increases for the PSTN depicted in Fig. 1. A well-known issue of MCTS, when limiting the number of chance node children (or also decision nodes in general), is that the first nodes (in terms of depth) of the tree impose a strong bias. A a consequence, the entire tree and therefore the estimated probability of success as $V(root)$ strongly depend on the sampled durations of t_2 and t_8. In fact, the plot shows two different runs of Lila, given 100 children (MCTS 100) at each chance node, for which each run converge to a somehow inaccurate approximation of the *NextFirst* robustness (which is of $\approx$11.35%). The approximation improves as the number of chance children increases to 1000, yet the strong bias is still visible in the plot. Finally, allowing an infinite number of chance children eventually correctly converges. Note that in this case, MCTS nodes at depth 4 are never visited, as the algorithm keeps always expanding at the same chance node for t_2. The progressive widening (MCTS PW) technique, which gradually grows the maximum number of children, have been here tested on chance nodes. Given adequate parameters, empirically set to $\beta = 0.3$ and $\alpha = 0.4$ in this experiment, Lila eventually converges accurately. These 700000+ iterations require approximately 10 seconds, on an Intel Core i7 2.3GHz, 16GB 3733MHz, CLang 12.0.

Approximating optimal dispatching. We now allow the number of children at decision nodes to grow as well, by using the progressive widening technique. This eventually enables MCTS to explore more elaborated decisions than just simply dispatch everything as soon as possible. Figure 5 shows how delaying the execution of rover driving activities, represented by time points t_1 and t_7, improves the estimated probability of success. Eventually, six different five-minute runs of Lila all converge to $\sim$48.5% chances of success, where t_1 should best be delayed to time 1$\sim$3 and t_7 to time 3$\sim$4, depending on the run.

Recall that under *NextFirst* dispatching protocol, the success probability was of $\approx$ 11.35% only. We clearly observe here that *NextFirst* produces significantly suboptimal dispatching decisions (by starting each activity as soon as possible), in the specific case of the PSTN at stake, which is the one depicted in Fig. 1.

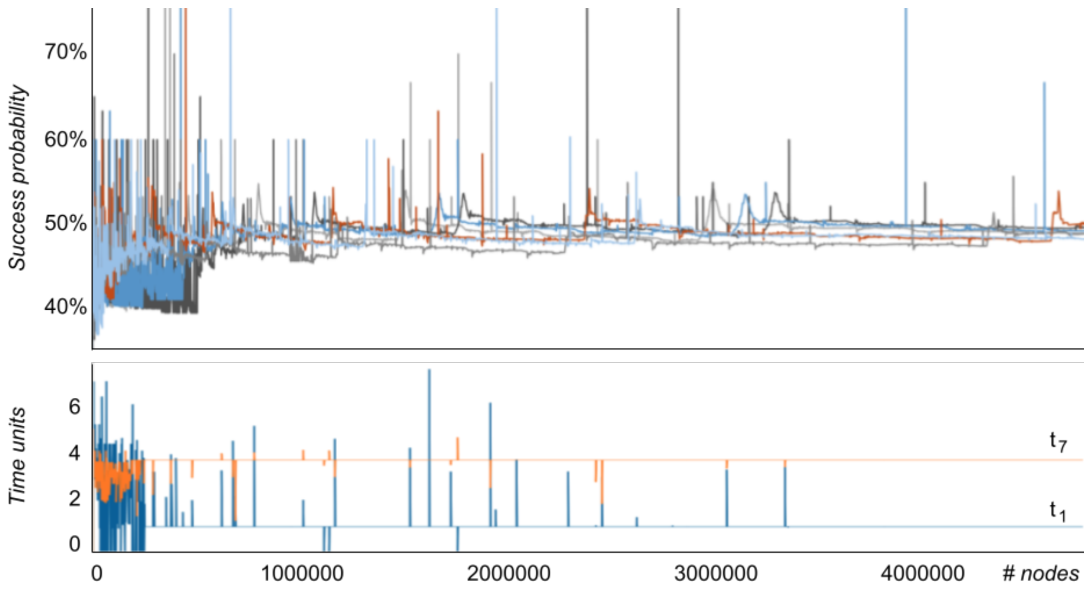

Figure 5: Top: Evolution of the $V(root)$ estimated root node value, when using progressive widening on both decision and chance nodes, for several independent runs of Lila. Bottom: Evolution of the best decisions as returned by the *root.BestChild()* function, for both t_1 and t_7, during one run of five minutes.

6 Conclusions and Future Work

We show how versatile the MCTS framework can be, when specialized to PSTN dispatching. It has the unlimited flexibility offered by the simulation paradigm. It is conjectured to asymptotically converges to optimal decisions (the demonstration is left for future work). In theory, it handles any possible continuous or discrete, even non-parametric, probability distributions, as well as inter-dependencies between random variables, exogenous as well as endogenous uncertainty. We conduct an preliminary experimental analysis, validating our algorithm, called *Lila*, on a simple PSTN example. Finally, we show how our easily algorithm may be adapted to deal with activity cutoff times and precondition chains (important in certain applications such as planetary rovers), hence providing a first solution framework to the problem of maximizing PSTN expected utility.

Future research directions.

Also related to the fact that our MCTS must deal with continuous domains, a *node.SelectChild()* function inspired from the theoretical results of Yee, Lisỳ, and Bowling 2016 may also be more appropriate than classical UCT. Finally, the literature already counts a number of improvements and extensions to classical MCTS (Browne et al. 2012), many of them being worth experimenting in the particular context of PSTN dispatching game. In addition to the aforementioned points, the PSTN extensions should be considered as well. In particular, accounting for activity resource usage, such as energy consumption, is of great interest in the context of Mars 2020 and future planetary rovers. In what follows we elaborate on additional promising directions.

Offline problem: DDC approximation. The experimental analysis conducted in this preliminary research is focused on only one PSTN instance, namely that of Fig. 1, allowing interesting insights on the algorithm behavior. A more comprehensive experimental study should be conducted on well known PSTN benchmarks, such as those recently considered for the *a priori* problem of approximating a PSTN robustness, or degree of dynamic controllability (DDC).

Online dispatching. This preliminary research includes an experimental analysis which considers the problem of evaluating the *a priori* PSTN robustness, but does not yet include online dispatching. A direct extension of this work is therefore to integrate *Lila* in an online dispatching simulator, in order to test its online reoptimization capabilities on well-known PSTN benchmarks. Since our algorithm is based on the MCTS framework which aims at dealing with fundamentally online problems, this should come with very little updates.

Proof of Concept: Mars 2020 planetary rover. How to define adequate cutoffs for M2020 task networks currently constitutes a real issue. We will i) evaluate the use of Lila to approximate the true DDC and expected utility of M2020 task networks, while considering cutoffs and precondition chains, and ii) will further try adapt the predefined cutoff times of some or all activities as part of the decisions, in order to maximize the success probability or the expected utility of the PSTNs. The latter (ii) is however a very hard problem. We will also consider iii) PSTN with resources (*e.g.* energy consumption) in addition to cutoffs and preconditions chains.

Acknowledgements

The research was partially carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).

References

Agrawal, J.; Chi, W.; Chien, S.; Rabideau, G.; Gaines, D.; and Kuhn, S. 2021a. Analyzing the effectiveness of rescheduling and Flexible Execution methods to address uncertainty in execution duration for a planetary rover. *Robotics and Autonomous Systems* 140: 103758.

Agrawal, J.; Chi, W.; Chien, S.; Rabideau, G.; Kuhn, S.; Gaines, D.; Vaquero, T.; and Bhaskaran, S. 2021b. Enabling limited resource-bounded disjunction in scheduling. *Journal of Aerospace Information Systems* 1–11.

Brooks, J.; Reed, E.; Gruver, A.; and Jr, J. C. B. 2015. Robustness in Probabilistic Temporal Planning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 3239–3246.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games* 4(1): 1–43.

Chaslot, G. M. J.; Winands, M. H.; HERIK, H. J. V. D.; Uiterwijk, J. W.; and Bouzy, B. 2008. Progressive strategies for Monte-Carlo tree search. *New Mathematics and Natural Computation* 4(03): 343–357.

Chi, W.; Agrawal, J.; Chien, S.; Fosse, E.; and Guduri, U. 2019. Optimizing Parameters for Uncertain Execution and Rescheduling Robustness. In *29th International Conference on Automated Planning and Scheduling (ICAPS)*.

Chi, W.; Chien, S.; Agrawal, J.; Rabideau, G.; Benowitz, E.; Gaines, D.; Fosse, E.; Kuhn, S.; and Biehl, J. 2018. Embedding a Scheduler in Execution for a Planetary Rover. In *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Couëtoux, A.; Hoock, J.-B.; Sokolovska, N.; Teytaud, O.; and Bonnard, N. 2011. Continuous upper confidence trees. In *International Conference on Learning and Intelligent Optimization*, 433–445. Springer.

Cowling, P. I.; Powley, E. J.; and Whitehouse, D. 2012. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games* 4(2): 120–143.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial intelligence* 49(1-3): 61–95.

Gaines, D.; Anderson, R.; Doran, G.; Huffman, W.; Justice, H.; Mackey, R.; Rabideau, G.; Vasavada, A.; Verma, V.; Estlin, T.; Fesq, L.; Ingham, M.; Maimone, M.; and Nesnas, I. 2016. Productivity Challenges for Mars Rover Operations. In *International Conference on Automated Planning and Scheduling, Planning and Robotics Workshop (PlanRob)*.

Kearns, M.; Mansour, Y.; and Ng, A. Y. 2002. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine learning* 49(2): 193–208.

Lund, K.; Dietrich, S.; Chow, S.; and Boerkoel, J. 2017. Robust execution of probabilistic temporal plans. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Melkó, E.; and Nagy, B. 2007. Optimal strategy in games with chance nodes. *Acta Cybernetica* 18(2): 171–192.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic Control of Plans with Temporal Uncertainty. In *International Joint Conference on Artificial Intelligence*, IJCAI'01, 494–499. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 1558608125.

Rabideau, G.; and Benowitz, E. 2017. Prototyping an Onboard Scheduler for the Mars 2020 Rover. *Proceedings of the International Workshop on Plaanning and Scheduling for Space, IWPSS* (Iwpss 2017).

Saint-Guillain, M.; Stegun Vaquero, T.; Agrawal, J.; and Chien, S. 2020. Robustness Computation of Dynamic Controllability in Probabilistic Temporal Networks with Ordinary Distributions. In Bessiere, C., ed., *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, 4168–4175. International Joint Conferences on Artificial Intelligence Organization. doi: 10.24963/ijcai.2020/576.

Saint-Guillain, M.; Stegun Vaquero, T.; Agrawal, J.; Chien, S.; and Abrahams, J. 2021. Probabilistic Temporal Networks with Ordinary Distributions: Theory, Robustness and Expected Utility. *Journal of Artificial Intelligence Research* .

Tsamardinos, I. 2002. A probabilistic approach to robust execution of temporal plans with uncertainty. In *Hellenic Conference on Artificial Intelligence*, 97–108. ISBN 978-3-540-43472-6. doi:10.1007/3-540-46014-4.

Yee, T.; Lisỳ, V.; and Bowling, M. H. 2016. Monte Carlo Tree Search in Continuous Action Spaces with Execution Uncertainty. In *IJCAI*, 690–697.